



Reinforcement learning for robotic manipulation using simulated locomotion demonstrations

Ozsel Kilinc¹ · Giovanni Montana¹

Received: 31 January 2021 / Revised: 4 August 2021 / Accepted: 25 October 2021
© The Author(s) 2021

Abstract

Mastering robotic manipulation skills through reinforcement learning (RL) typically requires the design of shaped reward functions. Recent developments in this area have demonstrated that using sparse rewards, i.e. rewarding the agent only when the task has been successfully completed, can lead to better policies. However, state-action space exploration is more difficult in this case. Recent RL approaches to learning with sparse rewards have leveraged high-quality human demonstrations for the task, but these can be costly, time consuming or even impossible to obtain. In this paper, we propose a novel and effective approach that does not require human demonstrations. We observe that every robotic manipulation task could be seen as involving a locomotion task from the perspective of the object being manipulated, i.e. the object could learn how to reach a target state on its own. In order to exploit this idea, we introduce a framework whereby an object locomotion policy is initially obtained using a realistic physics simulator. This policy is then used to generate auxiliary rewards, called simulated locomotion demonstration rewards (SLDRs), which enable us to learn the robot manipulation policy. The proposed approach has been evaluated on 13 tasks of increasing complexity, and can achieve higher success rate and faster learning rates compared to alternative algorithms. SLDRs are especially beneficial for tasks like multi-object stacking and non-rigid object manipulation.

Keywords Sparse rewards · Reinforcement learning · Robotic manipulation · Auxiliary rewards

Editors: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautmann

✉ Ozsel Kilinc
ozsel.kilinc@warwick.ac.uk

Giovanni Montana
g.montana@warwick.ac.uk

¹ WMG, University of Warwick, Coventry, UK

1 Introduction

Reinforcement learning (RL) solves sequential decision-making problems by learning a policy that maximises expected rewards. Recently, with the aid of deep artificial neural network as function approximators, RL-trained agents have been able to autonomously master a number of complex tasks, most notably playing video games (Mnih et al., 2015) and board games (Silver et al., 2016). Robot manipulation has been extensively studied in RL, but is particularly challenging to master because it often involves multiple stages (e.g. stacking multiple blocks), high-dimensional state spaces (e.g. dexterous hand manipulation Zhu et al., 2018; Andrychowicz et al., 2018) and complex dynamics (e.g. manipulating non-rigid objects). Although promising performance has been reported on a wide range of tasks like grasping (Levine et al., 2016; Popov et al., 2017), stacking (Nair et al., 2018a) and dexterous hand manipulation (Zhu et al., 2018; Andrychowicz et al., 2018), the learning algorithms usually require carefully-designed reward signals to learn good policies. For example, Popov et al. (2017) propose a thoroughly weighted 5-term reward formula for learning to stack Lego blocks and (Gu et al., 2017) use a 3-term shaped reward to perform door-opening tasks with a robot arm. The requirement of hand-engineered, dense reward functions limits the applicability of RL in real-world robot manipulation to cases where task-specific knowledge can be captured.

The alternative to designing shaped rewards consists of learning with only sparse feedback signals, i.e. a non-zero rewards indicating the completion of a task. Using sparse rewards is more desirable in practise as it generalises to many tasks without the need for hand-engineering (Silver et al., 2016; Andrychowicz et al., 2017; Burda et al., 2018a). On the other hand, learning with only sparse rewards is significantly more challenging since associating sequences of actions to non-zero rewards received only when a task has been successfully completed becomes more difficult. A number of existing approaches that address this problem have been proposed lately (Andrychowicz et al., 2017; Riedmiller et al., 2018; Houthoofd et al., 2016; Burda et al., 2018b; Pathak et al., 2017; Burda et al., 2018a; Savinov et al., 2018; Zhang et al., 2019); some of them report some success in completing manipulation tasks like object pushing (Andrychowicz et al., 2017; Zhang et al., 2019), pick-and-place (Andrychowicz et al., 2017), stacking two blocks (Riedmiller et al., 2018; Zhang et al., 2019), and target finding in a scene (Pathak et al., 2017; Savinov et al., 2018). Nevertheless, for more complex tasks such as stacking multiple blocks and manipulating non-rigid objects, there is scope for further improvement.

A particularly promising approach to facilitate learning has been to leverage human expertise through a number of manually generated examples demonstrating the robot actions required to complete a given task. When these demonstrations are available, they can be used by an agent in various ways, e.g. by attempting to generate a policy that mimics them (Ross et al., 2011; Bojarski et al., 2016; Xu et al., 2017), pre-learning a policy from them for further RL (Silver et al., 2016; Hester et al., 2018), as a mechanism to guide exploration (Nair et al., 2018a), as data from which to infer a reward function (Ng & Russell, 2000; Abbeel & Ng, 2004; Finn et al., 2016; Ho & Ermon, 2016), and in combination with trajectories generated during RL (Vecerik et al., 2017; Sasaki et al., 2019; Reddy et al., 2019). Practically, however, human demonstrations are expensive to obtain, and their effectiveness ultimately depends on the competence of the demonstrators. Demonstrators with insufficient task-specific expertise could generate low-quality demonstrations resulting in sub-optimal policies. Although there is an existing body of work focusing on learning with imperfect demonstrations (Grollman & Billard, 2011; Zheng et al., 2014; Shiarlis

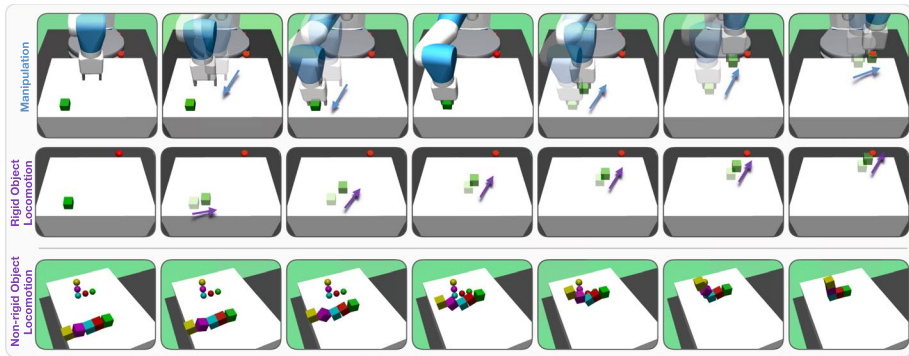


Fig. 1 An illustration of the proposed approach. Top row: a general robot manipulation task of pick-and-place, which requires the robot to pick up an object (green cube) and place it to a specified location (red sphere). Middle row: the corresponding auxiliary locomotion task requires the object to move to the target location. Bottom row: the auxiliary locomotion task corresponding to a pick-and-place task with a non-rigid object (not shown). Note that the auxiliary locomotion tasks usually have significantly simpler dynamics compared to the corresponding robot manipulation task, hence can be learnt efficiently through standard RL, even for very complex tasks. The learnt locomotion policy is used to inform the robot manipulation policy (Color figure online)

et al., 2016; Gao et al., 2018; Brown et al., 2019; Choi et al., 2019), these methods usually assume that either qualitative evaluation metrics are available (Grollman & Billard, 2011; Shiarlis et al., 2016; Brown et al., 2019) or that a substantial volume of demonstrations can be collected (Zheng et al., 2014; Gao et al., 2018; Choi et al., 2019).

In this paper, we propose a novel approach that allows complex robot manipulation tasks to be learnt with only sparse rewards. In the tasks we consider, an object is manipulated by a robot so that, starting from a (random) initial position, it eventually reaches a goal position through a sequence of states in which its location and pose vary. For example, Fig. 1(top row) represents a pick-and-place task in which the object is being picked up by the two-finger gripper and moved from its initial state to a pre-defined target location (red sphere). Our key observation is that every robot manipulation implies an underlying object locomotion task that can be explicitly modelled as an independent task for the object itself to learn. Figure 1(middle row) illustrates this idea for the pick-and-place task: the object, on its own, must learn to navigate from any given initial position until it reaches its target position. More complex manipulation tasks involving non-rigid objects can also be thought as inducing such object locomotion tasks; for instance, in Fig. 1(bottom row), a 5-tuple non-rigid object moves itself to the given target location and pose (see Fig. 3 for the description of the non-rigid object).

Although in the real world it is impossible for objects to move on their own, learning such object locomotion policies can be achieved in a virtual environment through a realistic physics engine such as MuJoCo (Todorov et al., 2012), Gazebo (Koenig & Howard, 2004) or Pybullet (Coumans & Bai, 2017). In our experience, such policies are relatively straightforward to learn using only sparse rewards since the objects usually operate in simple state/action spaces and/or have simple dynamics. Once a locomotion policy has been learnt, we utilise it to produce a form of auxiliary rewards guiding the main manipulation policy. We name these auxiliary rewards “Simulated Locomotion Demonstration Rewards” (SLDRs). During the process of learning the robot manipulation policy, the proposed SLDRs encourage the robot to execute policies implying object trajectories that are similar

to those obtained by the object locomotion policy. Although the SLDRs can only be learnt through a realistic simulator, this requirement does not restrict their applicability to real world problems, and the resulting manipulation policies can still be transferred to physical systems. To the best of our knowledge, this is the first time that object-level policies are trained in a physics simulator to enable robot manipulation learning driven by only sparse rewards.

In our implementation, all the policies are learnt using deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), which has been chosen due to its widely reported effectiveness in continuous control; however, most RL algorithms compatible with continuous actions could have been used within the proposed SLD framework. Our experimental results involve 13 continuous control environments using the MuJoCo physics engine (Todorov et al., 2012) within the OpenAI Gym framework (Brockman et al., 2016). These environments cover a variety of robot manipulation tasks with increasing level of complexity, e.g. pushing, sliding and pick-and-place tasks with a Fetch robotic arm, in-hand object manipulation with a Shadow's dexterous hand, multi-object stacking, and non-rigid object manipulation. Overall, across all environments, we have found that our approach can achieve faster learning rate and higher success rate compared to baselines methods, especially in more challenging tasks such as stacking objects and manipulating non-rigid objects. Baselines are provided to represent existing approaches that use reward-shaping, curiosity-based auxiliary rewarding and auxiliary goal generation techniques.

The remainder of the paper is organised as follows. In Sect. 2 we review the most related work, and in Sect. 3 we provide some introductory background material regarding the RL modelling framework and algorithms we use. In Sect. 4, we develop the proposed methodology. In Sect. 5 we describe all the environments used for our experiments, and the experimental results are reported in Sect. 6. Finally, we conclude with a discussion and suggestions for further extensions in Sect. 7.

2 Related work

2.1 Robotic manipulation

Robotics requires sequential decision making under uncertainty, and therefore it is a common application domain of machine learning approaches including RL (Kroemer et al., 2019). Recent advances in RL have focused on locomotion (Lillicrap et al., 2015; Schulman et al., 2015; Mnih et al., 2016) and manipulation tasks (Fu et al., 2016; Gu et al., 2017), which includes grasping (Levine et al., 2016; Popov et al., 2017), stacking (Nair et al., 2018a) and dexterous hand manipulation (Zhu et al., 2018; Andrychowicz et al., 2018). These tasks are particularly challenging as they require continuous control over actions and the expected behaviours are hard to formulate through rewards. Due to the sample inefficiency problem of RL, most state-of-the-art approaches rely on simulated environments such as MuJoCo (Todorov et al., 2012) as training using physical systems would be significantly slower and costly. Predicting how objects behave under manipulation has also been well studied. For example, Kopicki et al. (2009, 2011) and Belter et al. (2014) propose approaches to predict the motions of rigid objects under pushing actions with the aim of using these models to plan the robotic manipulation. Most recently, Li et al. (2018) has proposed to learn a particle-based dynamics from data to handle complex interactions between rigid bodies, deformable objects and fluids. The focus of these studies

has been to develop learnable simulators to replace traditional physics engines, whereas in this paper our aim is to learn object policies using the simulators. Although we employ a traditional physics engine for this paper, this could be replaced with learnable simulators in future work.

2.2 Learning from demonstrations

A substantial body of work exists on how to leverage such demonstrations, when available, for reinforcement learning. Behaviour cloning (BC) methods approach sequential decision-making as a supervised learning problem (Pomerleau, 1988; Duan et al., 2017; Bojarski et al., 2016; Xu et al., 2017). Some BC methods include an expert demonstrator in the training loop to handle the mismatching between the demonstration data and the data encountered in the training procedure (Ross et al., 2011; Ratliff et al., 2007). Recent BC methods have also considered adversarial frameworks to improve the policy learning (Ho & Ermon, 2016; Wang et al., 2017). A different approach consists of inverse reinforcement learning, which seeks to infer a reward/cost function to guide the policy learning (Ng & Russell, 2000; Abbeel & Ng, 2004; Finn et al., 2016). Several methods have been developed to leverage demonstrations for robotic manipulation tasks with sparse rewards. For instance, Vecerik et al (2017) and Nair et al. (2018a) jointly use demonstrations with trajectories collected during the RL process to guide the exploration, and Hester et al. (2018) use the demonstrations to pre-learn a policy, which is further fine-tuned in a following RL stage. Obtaining the training data requires specialised data capture setups such as teleoperation interfaces. In general, obtaining good quality demonstrations is an expensive process in terms of both human effort and equipment requirements. In contrast, the proposed method generates object-level demonstrations autonomously, and could potentially be used jointly with human-generated demonstrations when these are available.

2.3 Goal conditioned policies and auxiliary goal generation

Goal-conditioned policies (Schaul et al., 2015) that can generalise over multiple goals have been shown to be promising for robotic problems. For manipulation tasks with sparse rewards, several approaches have recently been proposed to automatically generate the auxiliary goals. For instance, Sukhbaatar et al. (2018) used a self-play approach on reversible or resettable environments, Florensa et al. (2018) employed adversarial training for robotic locomotion tasks, Nair et al. (2018b) proposed variational autoencoders for visual robotics tasks, and Andrychowicz et al. (2017) introduced Hindsight Experience Replay (HER), which randomly draws synthetic goals from previously encountered states. HER in particular has been proved particularly effective, although the automatic goal generation can still be problematic on complex tasks involving multiple stages, e.g. stacking multiple objects, when used without demonstrations (Nair et al., 2018b). Some attempts have been made to form an explicit curriculum for such complex tasks; e.g. Riedmiller et al. (2018) manually define several semantically grounded sub-tasks each having its own individual reward. Methods such as this one requires significant human effort hence they cannot be readily applied across different tasks. The proposed method in this paper uses goal-conditioned policies and adopts HER for auxiliary goal generation due to its effectiveness in robotic manipulation. However, it can be integrated with the other goal techniques in the literature.

2.4 Auxiliary rewards in RL

Lately, increasing efforts have been made to design general auxiliary reward functions aimed at facilitating learning in environments with only sparse rewards. Many of these strategies involve a notion of curiosity (Schmidhuber, 1991), which encourages agents to visit novel states that have not been seen in previous experience; for instance, Pathak et al. (2017) formulate the auxiliary reward using the error in predicting the RL agent's actions by an inverse dynamics model, Houthoofd et al. (2016) encourage the agent to visit the states that result the largest information gain in system dynamics, Burda et al. (2018a) construct the auxiliary reward based on the error in predicting the output of a fixed randomly initialised neural network, and Savinov et al. (2018) introduces the notion of state reachability. Despite the benefits introduced by these approaches, visiting unseen states may be less beneficial in robot manipulation tasks as exploring complex state spaces to find rewards is rather impractical (Andrychowicz et al., 2017). The proposed approach, on the other hand, produces the auxiliary rewards based on the underlying object locomotion; as such, it motivates the robot to mimic the optimal object locomotion rather than curiously exploring the continuous state space.

3 Background

3.1 Multi-goal RL for robotic manipulation

We are concerned with solving a manipulation task: an object is presented to the robot, and has to be manipulated so as to reach a target position. In the tasks we consider, the target goal is specified by the object location and orientation, and the robot is rewarded only when it reaches its goal. We model the robot's sequential decision process as a Markov Decision Process (MDP) defined by a tuple, $M = \langle \mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where \mathcal{S} is the set of states, \mathcal{G} is the set of goals, \mathcal{A} is the set of actions, \mathcal{T} is the state transition function, \mathcal{R} is the reward function and $\gamma \in [0, 1)$ is the discounting factor. At the beginning of an episode, the environment samples a goal $g \in \mathcal{G}$. The state of the environment at time t is denoted by $s_t \in \mathcal{S}$ and includes both robot-related and object-related features. In a real system, these features are typically continuous variables obtained through sensors of the robot. The position of the object o_t is one of the object-related features included in s_t and can be obtained through a known mapping, i.e. $o_t = m_{\mathcal{S} \rightarrow \mathcal{O}}(s_t)$. A robot's action is controlled by a deterministic policy, i.e. $a_t = \mu_\theta(s_t, g) : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, parameterised by θ . The environment moves to its next state through its state transition function, i.e. $s_{t+1} = \mathcal{T}(s_t, a_t) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and provides an immediate and sparse reward r_t , defined as

$$r_t = \mathcal{R}(o_{t+1}, g) = \begin{cases} 0, & \text{if } \|o_{t+1} - g\|_2 \leq \epsilon \\ -1, & \text{otherwise} \end{cases} \quad (1)$$

where ϵ is a pre-defined threshold. Following its policy, the robot interacts with the environment until the episode terminates after T steps. The interaction between the robot and the environment generates a trajectory, $\tau = (g, s_1, a_1, r_1, \dots, s_T, a_T, r_T, s_{T+1})$. The ultimate learning objective is to find the optimal policy that maximises the expected sum of the discounted rewards over the time horizon T , i.e.

$$J(\mu_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\tau|\mu_\theta)} \left[R(\tau) = \sum_{i=1}^T \gamma^{i-1} r_i \right] \quad (2)$$

where γ is the discount factor.

3.2 Deep deterministic policy gradient algorithm

Policy gradient (PG) algorithms update the policy parameters θ in the direction of $\nabla_\theta J(\mu_\theta)$ to maximise the expected return $J(\mu_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\tau|\mu_\theta)} [R(\tau)]$. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) integrates non-linear function approximators such as neural networks with Deterministic Policy Gradient (DPG) (Silver et al., 2014) that uses deterministic policy functions. DDPG maintains a policy (actor) network $\mu_\theta(s_t, g)$ and an action-value (critic) network $Q^\mu(s_t, a_t, g)$.

The actor $\mu_\theta(s_t, g)$ deterministically maps states to actions. The critic $Q^\mu(s_t, a_t, g)$ estimates the expected return when starting from s_t by taking a_t , and then following μ_θ in the future states until the termination of the episode, i.e. $Q^\mu(s_t, a_t, g) = \mathbb{E} \left[\sum_{i=t}^T \gamma^{i-t} r_i | s_t, a_t, g, \mu_\theta \right]$. When interacting with the environment, DDPG assures the exploration by adding a noise to the deterministic policy output, i.e. $a_t = \mu_\theta(s_t, g) + \mathcal{N}$. Experienced transitions during these interactions, i.e. $\langle g, s_t, a_t, r_t, s_{t+1} \rangle$, are stored in a replay buffer \mathcal{D} . The actor and critic networks are updated using the transitions sampled from \mathcal{D} . The critic parameters are learnt by minimising the following loss to satisfy the Bellman equation similarly to Q-learning (Watkins and Dayan 1992):

$$\mathcal{L}(Q^\mu) = \mathbb{E}_{g, s_t, a_t, r_t, s_{t+1} \sim \mathcal{D}} \left[(Q^\mu(s_t, a_t, g) - y)^2 \right] \quad (3)$$

where $y = r_t + \gamma Q^\mu(s_t, \mu(s_{t+1}), g)$. The actor parameters θ are updated using the following policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{g, s_t \sim \mathcal{D}} \left[\nabla_a Q^\mu(s_t, a, g) |_{a=\mu_\theta(s_t, g)} \nabla_\theta \mu_\theta(s_t, g) \right] \quad (4)$$

We adopt DDPG as the main training algorithm; however, the proposed idea can also be used with other off-policy approaches that work with continuous action domains.

3.3 Hindsight experience replay

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) has been introduced to learn policies from sparse rewards, especially for robot manipulation tasks. The idea is to view the states achieved in an episode as pseudo goals (i.e. achieved goals) to facilitate learning even when the desired goal has not been achieved during the episode. Suppose we are given an observed trajectory, $\tau = (g, s_1, a_1, r_1, \dots, s_T, a_T, r_T, s_{T+1})$. Since o_t can be obtained from s_t using a fixed and known mapping, the path that was followed by the object during the trajectory, i.e. o_1, \dots, o_{T+1} , can be easily extracted. HER samples a new goal from this path, i.e. $\tilde{g} \sim \{o_1, \dots, o_T\}$, and the rewards are recomputed with respect to \tilde{g} , i.e. $\tilde{r}_t = \mathcal{R}(o_{t+1}, \tilde{g})$. Using these rewards and \tilde{g} , a new trajectory is created implicitly, i.e. $\tilde{\tau} = (\tilde{g}, s_1, a_1, \tilde{r}_1, \dots, s_T, a_T, \tilde{r}_T, s_{T+1})$. These HER trajectories $\tilde{\tau}$ are used to train the policy parameters together with the original trajectories.

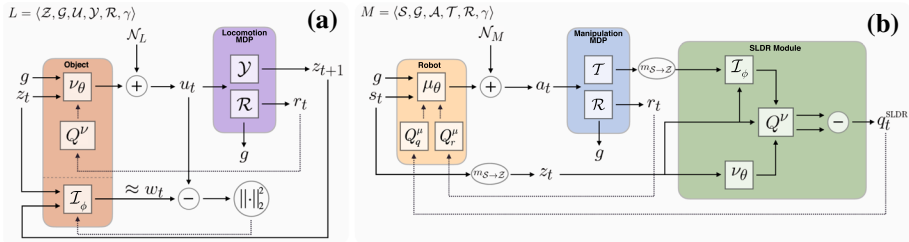


Fig. 2 Block diagram for the proposed SLD algorithm. The solid lines represent the forward pass of the model. The dashed lines represent the rewards/losses as feedback signals for model learning

4 Methodology

Given a manipulation task, initially we introduce a corresponding auxiliary locomotion task for the object that is being manipulated, i.e. the object is assumed to be the decision-making agent. This auxiliary problem is usually significantly easier to learn compared to the original task. After learning the object locomotion policy, we use it on a reward-generating mechanism for the robot when learning the original manipulation task. In this section, we explain the steps involved in our proposed procedure, i.e. (a) how the object locomotion policies are learned, (b) how the proposed reward function is defined, and (c) how these auxiliary rewards are leveraged for robotic manipulation.

4.1 Object locomotion policies

The object involved in the manipulation task is initially modelled as an agent capable of independent decision making abilities, and its decision process is modelled by a separate MDP defined by a tuple $L = \langle \mathcal{Z}, \mathcal{G}, \mathcal{U}, \mathcal{Y}, \mathcal{R}, \gamma \rangle$. Here, \mathcal{Z} is the set of states, \mathcal{G} is the set of goals, \mathcal{U} is the set of actions, \mathcal{Y} is the state transition function, \mathcal{R} is the reward function and $\gamma \in [0, 1)$ is the discounting factor. The same goal space, \mathcal{G} , is used as in M , and $z_t \in \mathcal{Z}$ is a reduced version of s_t that only involves object-related features including the position of the object, i.e. $o_t \subset z_t$. The object's action space explicitly controls the pose of the object, and these actions are controlled by a deterministic policy, i.e. $u_t = \nu_\theta(z_t, g) : \mathcal{Z} \times \mathcal{G} \rightarrow \mathcal{U}$. The state transition is defined on a different space, i.e. $\mathcal{Y} : \mathcal{Z} \times \mathcal{U} \rightarrow \mathcal{Z}$; however, the same sparse reward function is used here as before. Figure 2a illustrates the training procedure used in this context and based on DDPG with HER. The optimal object policy ν_θ maximises the expected return $J(\nu_\theta) = \mathbb{E}_{g, z_t \sim \mathcal{D}_L} [\sum_{i=1}^T \gamma^{i-1} r_i]$ where \mathcal{D}_L denotes the replay buffer containing the trajectories, indicated by η , obtained by ν_θ throughout training.

4.2 Robotic manipulation with simulated locomotion demonstration rewards (SLDR)

On the original manipulation task M , the robot receives the current environmental state and the desired goal and then decides how to act according to its policy μ_θ . Whenever the object is moved from one position to another, the observed object locomotion is a consequence of robot's actions. More concretely, the observed object action on M (hereafter denoted by w_t)

is a function of the robot policy μ_θ . The relation between w_t and μ_θ depends on the environmental dynamics whose close-form model is unknown. We use $f : \mathcal{A} \rightarrow \mathcal{U}$ to denote this unknown relation, i.e. $w_t = f(\mu_\theta(s_t, g))$.

The key steps of the proposed approach are as follows: as we had initially learnt an object locomotion policy on L , first we use it to enquire the optimal object action for the current state and goal, i.e. $u_t = v_\theta(z_t, g)$. Then, we update μ_θ in order to make w_t get closer to u_t . This learning objective can be written as follows:

$$\arg \min_{\mu_\theta} \mathbb{E}_{s_t} \|w_t - u_t\|_2^2 = \arg \min_{\mu_\theta} \mathbb{E}_{s_t \sim P(s_t | \mu_\theta)} \|f(\mu_\theta(s_t, g)) - v_\theta(z_t, g)\|_2^2 \quad (5)$$

Given that the the environment dynamics is unknown, we replace f in Eq. (5) with a parameterised model to approximate w_t . Estimating w_t from robot actions is not straight-forward as it requires keeping track of all previous actions, i.e. $a_{1:t}$, and the initial state. Instead, we propose to estimate w_t by evaluating the transition from the current state to the next. Specifically, we substitute f with a parameterised inverse dynamic model, i.e. $\mathcal{I}_\phi : \mathcal{Z} \times \mathcal{Z} \rightarrow \mathcal{U}$, that we train to estimate the output of $v_\theta(z_t, g)$ from z_t and z_{t+1} , i.e. $v_\theta(z_t, g) \approx \mathcal{I}_\phi(z_t, z_{t+1})$. We learn the parameters of \mathcal{I}_ϕ on the object locomotion task L (see Sect. 4.3 and Algorithm 1 for training details), and then employ the trained model on the manipulation task M to approximate w_t . Substituting \mathcal{I}_ϕ into Eq. (5) leads to the following optimisation problem:

$$\arg \min_{\mu_\theta} \mathbb{E}_{s_t} \|w_t - u_t\|_2^2 \approx \arg \min_{\mu_\theta} \mathbb{E}_{s_t \sim P(s_t | \mu_\theta)} \|\mathcal{I}_\phi(z_t, z_{t+1}) - v_\theta(z_t, g)\|_2^2 \quad (6)$$

On M , z_{t+1} is a function of $\mathcal{T}(s_t, \mu_\theta(s_t, g))$. In our setting, the close-form of the state transition function \mathcal{T} is unknown, instead \mathcal{T} can only be sampled. Also, pursuing a model-free approach, we do not aim to learn a model for \mathcal{T} . Therefore, minimising Eq. (6) through gradient-based methods is not an option for our setting as this would require differentiation through \mathcal{T} . Instead, we propose to formalise this objective as a reward to be maximised through a standard model-free RL approach. The first obvious candidate for this reward notion can be written as follows:

$$q_t = -\|w_t - u_t\|_2^2 \approx -\|\mathcal{I}_\phi(z_t, z_{t+1}) - v_\theta(z_t, g)\|_2^2 \quad (7)$$

Practically, however, the above reward is sensitive to the scales of $\mathcal{I}_\phi(z_t, z_{t+1})$ and $v_\theta(z_t, g)$, and therefore it may require an additional normalisation term. Even with a normalisation term, the scale of the rewards would shift throughout the training depending on the exploration and the sampling. In order to deal with this issue, we propose another reward notion adopting Q^ν , i.e. the action-value function that had been learnt for v_θ on object locomotion task L (see Sect. 4.3 and Algorithm 1 for training details). The proposed reward notion is written as follows:

$$\begin{aligned} q_t^{\text{SLDR}} &= Q^\nu(z_t, w_t, g) - Q^\nu(z_t, u_t, g) \\ &\approx Q^\nu(z_t, \mathcal{I}_\phi(z_t, z_{t+1}), g) - Q^\nu(z_t, v_\theta(z_t, g), g) \end{aligned} \quad (8)$$

We refer to Eq. (8) as the Simulated Locomotion Demonstrations Rewards (SLDR). Rather than comparing w_t and u_t directly with each other as in Eq. (7), the SLDR compares their action-values using Q^ν . Being learnt on L using sparse rewards, Q^ν is well-bounded (Sutton and Barto 2018), and q_t^{SLDR} produced adopting Q^ν does not require a normalisation term.

Note that, by definition, $Q^v(z_t, u, g)$ gives the expected return for any object locomotion action $u \in \mathcal{U}$, when it is taken at the current state z_t and then v_θ is followed for the future states. Since v_θ had been learnt through standard RL to maximise the sparse rewards, it is the optimal object locomotion policy, and therefore $Q^v(z_t, v_\theta(z_t, g), g)$ gives the maximum expected return. Accordingly, q_t^{SLDR} can be viewed as the advantage of w_t with respect to $v_\theta(z_t, g)$ in terms of the action-values, and is expected to be non-positive. Maximising this term encourages the robot to induce similar object actions compared to the optimal ones according to v_θ .

Algorithm 1: Learning locomotion policy and inverse dynamic

Given : Locomotion MDP $L = \langle \mathcal{Z}, \mathcal{G}, \mathcal{U}, \mathcal{Y}, \mathcal{R}, \gamma \rangle$,
 Neural networks v_θ , Q^v and \mathcal{I}_ϕ
 A random process \mathcal{N}_L for exploration
 Fixed and known mapping function $m_{\mathcal{Z} \rightarrow \mathcal{O}} : \mathcal{Z} \rightarrow \mathcal{O}$
Initialise: Parameters of v_θ , Q^v and \mathcal{I}_ϕ
 Experience replay buffer \mathcal{D}_L

for $i_{\text{episode}} = 1$ **to** N_{episode} **do**
 for $i_{\text{rollout}} = 1$ **to** N_{rollout} **do**
 Receive initial state z_1 and g , $o_1 = m_{\mathcal{Z} \rightarrow \mathcal{O}}(z_1)$
 for $t = 1, T$ **do**
 Sample an object action: $u_t = v_\theta(z_t, g) + \mathcal{N}_L$
 Execute the action: $z_{t+1} = \mathcal{Y}(z_t, a_t)$, $r_t = \mathcal{R}(o_{t+1}, g)$
 Store $(g, z_t, u_t, r_t, z_{t+1})$ in \mathcal{D}_L
 Generate HER samples and store in \mathcal{D}_L
 for $i_{\text{update}} \leftarrow 1$ **to** N_{update} **do**
 Get a random mini-batch of samples from \mathcal{D}_L
 Update Q^v minimising the loss in Eq.(3)
 Update v_θ using the gradient in Eq.(4)
 Update \mathcal{I}_ϕ minimising the loss in Eq.(9)

Return : v_θ , Q^v and \mathcal{I}_ϕ

4.3 Learning algorithms

In this subsection, we detail the learning algorithms for the object locomotion and the robotic manipulation policies. Figure 2 shows the block diagrams of the learning procedures.

4.3.1 Object locomotion policy

We learn the object locomotion policy only using the environmental sparse rewards as described in Algorithm 1. We adopt DDPG (Sect. 3.2) as the training framework together with HER (Sect. 3.3) to generate auxiliary transition samples to deal with the exploration difficulty caused by the sparse rewards. Q^v is updated to minimise Eq. (3), and v_θ is optimised using the gradient in Eq. (4). Concurrently, we learn \mathcal{I}_ϕ using the trajectories generated during the policy learning process by minimising the following objective function:

$$\arg \min_{\phi} \mathbb{E}_{z_t, u_t, z_{t+1} \sim \mathcal{D}_L} \left\| \mathcal{I}_\phi(z_t, z_{t+1}) - u_t \right\|_2^2 \quad (9)$$

where \mathcal{D}_L is an experience replay buffer.

4.3.2 Robotic manipulation policy

Similarly, we learn the robotic manipulation policy adopting DDPG with HER as described in Algorithm 2. Using the optimisation objective given in Eq. (3), we learn two

action-value functions: Q_r^μ for the environmental sparse rewards r_t , and Q_q^μ for the proposed SLDR q_t^{SLDR} . Accordingly, μ_θ is updated following the gradient below that uses both action-value functions:

$$\nabla_\theta J(\theta) = \mathbb{E}_{g, s_t \sim \mathcal{D}_M} \left[\nabla_a (Q_r^\mu(s_t, a, g) + Q_q^\mu(s_t, a, g)) \Big|_{a=\mu(s_t, g)} \nabla_\theta \mu_\theta(s_t, g) \right] \quad (10)$$

where \mathcal{D}_M is an experience replay buffer. Some tasks may include $N > 1$ objects, e.g. stacking. The proposed method is able to handle these tasks by using individual SLDR for each object and learning individual $Q_{q_i}^\mu$ for each one of them. Then, the gradient required to update μ_θ is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{g, s_t \sim \mathcal{D}_M} \left[\nabla_a \left(Q_r^\mu(s_t, a, g) + \sum Q_{q_i}^\mu(s_t, a, g) \right) \Big|_{a=\mu(s_t, g)} \nabla_\theta \mu_\theta(s_t, g) \right] \quad (11)$$

Algorithm 2: Learning manipulation policy

Given : Manipulation MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$,
Learnt object-related components ν_θ , Q^ν and \mathcal{I}_ϕ
Neural networks μ_θ , Q_r^μ and Q_q^μ
A random process \mathcal{N}_M for exploration
Fixed and known mapping function $m_{\mathcal{S} \rightarrow \mathcal{O}} : \mathcal{S} \rightarrow \mathcal{O}$
Fixed and known mapping function $m_{\mathcal{S} \rightarrow \mathcal{Z}} : \mathcal{S} \rightarrow \mathcal{Z}$

Initialise: Parameters of μ_θ , Q_r^μ and Q_q^μ
Experience replay buffer \mathcal{D}_M

for $i_{\text{episode}} = 1$ **to** N_{episode} **do**
 for $i_{\text{rollout}} = 1$ **to** N_{rollout} **do**
 Receive initial state s_1 and g , $o_1 = m_{\mathcal{S} \rightarrow \mathcal{O}}(s_1)$
 for $t = 1, T$ **do**
 Sample a robot action: $a_t = \mu_\theta(s_t, g) + \mathcal{N}_M$
 Execute the action: $s_{t+1} = \mathcal{T}(s_t, a_t)$, $r_t = \mathcal{R}(o_{t+1}, g)$
 Obtain SLD reward:
 $z_t = m_{\mathcal{S} \rightarrow \mathcal{Z}}(s_t)$ and $z_{t+1} = m_{\mathcal{S} \rightarrow \mathcal{Z}}(s_{t+1})$
 $q_t^{\text{SLDR}} = Q^\nu(z_t, \mathcal{I}_\phi(z_t, z_{t+1}), g) - Q^\nu(z_t, \nu_\theta(z_t, g), g)$
 Store $(g, s_t, a_t, r_t, q_t, s_{t+1})$ in \mathcal{D}_M
 Generate HER samples and store in \mathcal{D}_M

for $i_{\text{update}} \leftarrow 1$ **to** N_{update} **do**
 Get a random mini-batch of samples from \mathcal{D}_M
 Update Q_r^μ minimising the loss in Eq.(3) for r_t
 Update Q_q^μ minimising the loss in Eq.(3) for q_t^{SLDR}
 Update μ_θ using the gradient in Eq.(10)

Return : μ

5 Environments

We have evaluated the SLD method on 13 simulated MuJoCo (Todorov et al., 2012) environments using two different robot configurations: 7-DoF Fetch robotic arm with a two-finger parallel gripper and 24-DoF Shadow's Dexterous Hand. The tasks we have chosen to evaluate include single rigid object manipulation, multiple rigid object stacking and non-rigid object manipulation. Overall, we have used 9 MuJoCo environments (3 with Fetch robot arm and 6 with Shadow's hand) for single rigid object tasks. Furthermore, we have

included additional environments for multiple object stacking and non-rigid object manipulation using the Fetch robot arm. In all environments the rewards are sparse.

5.1 Fetch arm single object environments

These are the same *Push*, *Slide* and *PickAndPlace* tasks introduced in Plappert et al. (2018). In each episode, a desired 3D position (i.e. the target) of the object is randomly generated. The reward is zero if the object is within 5 cm range to the target, otherwise -1 . The robot actions are 4-dimensional: 3D for the desired arm movement in Cartesian coordinates and 1D to control the opening of the gripper. In pushing and sliding, the gripper is locked to prevent grasping. The observations include the positions and linear velocities of the robot arm and the gripper, the object's position, rotation, angular velocity, the object's relative position and linear velocity to the gripper, and the target coordinate. An episode terminates after 50 time-steps.

5.2 Shadow's hand single object environments

These include the tasks first introduced in Plappert et al. (2018), i.e. *Egg*, *Block*, *Pen* manipulation. In these tasks, the object (a block, an egg-shaped object, or a pen) is placed on the palm of the robot hand; the robot hand is required to manipulate the object to reach a target pose. The target pose is 7D describing the 3D position together with 4D quaternion orientation, and is randomly generated in each episode. The reward is 0 if the object is within some task-specific range to the target, otherwise -1 . As in Plappert et al. (2018), each task has two variants: *Full* and *Rotate*. In the *Full* variant, the object's whole 7D pose is required to meet the given target pose. In the *Rotate* variants, the 3D object position is ignored and only the 4D object rotation is expected to satisfy the desired target. Robot actions are 20-dimensional controlling the absolute positions of all non-coupled joints of the hand. The observations include the positions and velocities of all 24 joints of the robot hand, the object's position and rotation, the object's linear and angular velocities, and the target pose. An episode terminates after 100 time-steps.

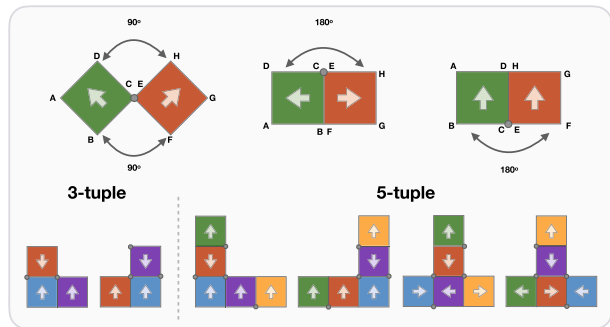
5.3 Fetch arm multiple object stacking environments

The stacking task is built upon the *PickAndPlace* task. We consider 2- and 3-object stacking tasks. For N -object stacking task, the target has $3N$ dimensions describing the desired positions of all N objects in 3D. Following Nair et al. (2018a), we start these tasks with the first object placed at its desired target. The robot needs to perform $N - 1$ pick-and-place actions without displacing the first object. The reward is zero if all objects are within 5cm range to their designated targets, otherwise the reward is assigned a value of -1 . The robot actions and observations are similar to those in the *PickAndPlace* task. The episode length is 50 time-steps for 2-object stacking and 100 for 3-object.

5.4 Fetch arm non-rigid object environments

We build non-rigid object manipulation tasks based on the *PickAndPlace* task. Instead of using the original rigid block, we have created a non-rigid object by hinging some blocks side-by-side along their edges as shown in Fig. 3. A hinge joint is placed between two

Fig. 3 An illustration of the non-rigid objects used in the experiments. Top row: a hinge joint (shown as grey circles) between two neighbouring blocks allows one rotational DoF along their coincident edges up to 180° . Bottom row: each variant has some predefined target poses (2 options for *3-tuple* and 4 for *5-tuple*)



neighbouring blocks, allowing one rotational degree of freedom (DoF) along their coincident edges up to 180° . We introduce two different variants: *3-tuple* and *5-tuple*. For the N -tuple task, N cubical blocks are connected with $N - 1$ hinge joints creating $N - 1$ internal DoF. The target pose has $3N$ -dimension describing the desired 3D positions of all N blocks, which are selected uniformly in each episode from a set of predefined target poses (see Fig. 3). The robot is required to manipulate the object to match the target pose. The reward is zero when all the N blocks are within a 2cm range to their corresponding targets, otherwise -1 . Robot actions and observations are similar to those in the *PickAnd-Place* tasks, excepting that the observations include the position, rotation, angular velocity, relative position and linear velocity to the gripper for each block. The episode length is 50 time-steps for both *3-tuple* and *5-tuple*.

5.5 Object locomotion environments

For each robotic manipulation task described above, we use an object locomotion task where we first learn v_θ , Q^v and \mathcal{I}_ϕ . Here, we detail the observation and action space differences between object locomotion and robotic manipulation tasks.

For any task, the object's observation is a subset of the robot's observation, i.e. $z_t \subset s_t$, and only includes object-related features while excluding those related to the robot. More concretely, for the environments with the Fetch arm, the object's observations include the object's position, rotation, angular velocity, the object's relative position and linear velocity to the target, and the target location. For the environments with the Shadow's hand, the object observations include the object's position and rotation, the object's linear and angular velocities, and the target pose. We define the object action as the desired relative change in the 7D object pose (3D position and 4D quaternion orientation) between two consecutive time-steps. This leads to 7D action spaces. Specifically for non-rigid objects, we define the object action as the desired relative change in the poses of the blocks at two ends. This leads to 14D action spaces. The rewards are the same as those in each robot manipulation task.

It is worth noting that, in the *Full* variants of Shadow's hand environments, we consider the object translation and rotation as two individual locomotion tasks, and we learn separate locomotion policies and Q-functions for each task. We find that the above strategy encourages the manipulation policy to perform translation and rotation simultaneously. Although object translation and rotation could be executed within a single task, we have empirically found that the resulting manipulation policies tend to prioritise one behaviour

versus the other (e.g. they tend to rotate the object first, then translate it) and generally achieves a lower performance.

6 Experiments

6.1 Implementation and training process

Three-layer neural networks with ReLU activations was used to approximate all policies, action-value functions and inverse dynamics models. The Adam optimiser (Kingma and Ba 2014) was employed to train all the neural networks. During the training of locomotion policies, the robot was considered as a non-learning component in the scene and its actions were not restricted to prevent any potential collision with the objects. We could have different choices for the actions of the robot. For example, we could let the robot move randomly or perform any arbitrary fixed action (e.g. a robot arm moving upwards with constant velocity until it reaches to the maximum height and then staying there). In preliminary experiments, we assessed whether this choice bears any effect on final performance, and concluded that no particular setting had clear advantages. For learning locomotion and manipulation policies, most of the hyperparameters suggested in the original HER implementation (Plappert et al., 2018) were retained with only a couple of exceptions for locomotion policies only: to facilitate exploration, with probability 0.2 (0.3 in Plappert et al., 2018) a random action was drawn from a uniform distribution, otherwise we retained the current action, and added Gaussian noise with zero mean and 0.05 (0.2 in Plappert et al., 2018) standard deviation. For locomotion policies, in all Shadow's hand environments and *5-tuple*, we train the objects over 50 epochs. In the remaining environments, we stop the training after 20 epochs. When training the main manipulation policies, the number of epochs varies across tasks. For both locomotion and manipulation policies, each epoch includes 50 cycles, and each cycle includes 38 rollouts generated in parallel through 38 MPI workers using CPU cores. This leads to $38 \times 50 = 1900$ full episodes per epoch. For each epoch, the parameters are updated 40 times using a batch size of 4864 on a GPU core. We normalise the observations to have zero mean and unit standard deviation as input of neural networks. We update mean and standard deviations of the observations using running estimation on the data in each rollout. We clip q_t^{SLDR} to the same range with the environmental sparse rewards, i.e. $[-1, 0]$.

Our algorithm has been implemented in PyTorch.¹ All the environments are based on OpenAI Gym. The corresponding source code, the environments, and illustrative videos for selected tasks have been made publicly available.²³⁴

6.2 Comparison and performance evaluation

We include the following methods for comparisons:

¹ <https://pytorch.org/>.

² Source code: <https://github.com/WMGDataScience/sldr.git>.

³ Environments: https://github.com/WMGDataScience/gym_wmgds.git.

⁴ Supplementary videos: <https://youtu.be/jubZ0dPVl2M>.

- *DDPG-Sparse* Refers to DDPG (Lillicrap et al., 2015) using sparse rewards.
- *HER-Sparse* Refers to DDPG with HER (Andrychowicz et al., 2017) using sparse rewards.
- *CHER-Sparse* Refers to DDPG with Curriculum-guided Hindsight Experience Replay (Fang et al., 2019) using sparse rewards.
- *HER-Dense* Refers to DDPG with HER, using dense distance-based rewards.
- *DDPG-Sparse+SLDR* Refers to DDPG using sparse environmental rewards and SLDR proposed in this paper.
- *HER-Sparse+RNDR* Refers to DDPG with HER, using sparse environmental rewards and random network distillation-based auxiliary rewards (RNDR) (Burda et al., 2018a).
- *HER-Sparse+SLDR* Refers to DDPG with HER, using sparse environmental rewards and SLDR.

We use DDPG-Sparse, HER-Sparse and HER-Dense as baselines. HER-Sparse+RNDR is a representative method constructing auxiliary rewards to facilitate policy learning. CHER-Sparse replaces the random selection mechanism of HER with an adaptive one that considers the proximity to true goals. DDPG-Sparse+SLDR and HER-Sparse+SLDR represents the proposed approach using SLDR with different methods for policy learning.

Following Plappert et al. (2018), we evaluate the performance after each training epoch by performing 10 deterministic test rollouts for each one of the 38 MPI workers. Then we compute the test success rate by averaging across the 380 test rollouts. For all comparison methods, we evaluate the performance with 5 different random seeds and report the median test success rate with the interquartile range. In all environments, we also keep the models with the highest test success rate for different methods and compare their performance.

6.3 Single rigid object environments

The learning curves for Fetch, the *Rotate* and *Full* variants of Shadow's hand environments are reported in Figs. 4a and 5a, b, respectively. We find that HER-Sparse+SLDR features a faster learning rate and the best performance on all the tasks. This evidence demonstrates that SLDR, coupled with DDPG and HER, can facilitate policy learning with sparse rewards. The benefits introduced by HER-Sparse+SLDR are particularly evident in hand manipulation tasks (Fig. 5a, b) compared to fetch robot tasks (Fig. 4a), which are notoriously more complex to solve. Additionally, we find that HER-Sparse+SLDR outperforms HER-Sparse+RNDR in most tasks. A possible reason for this result is that most methods using auxiliary rewards are based on the notion of curiosity, whereby reaching unseen states is a preferable strategy, which is less suitable for manipulation tasks (Andrychowicz et al., 2017). In contrast, the proposed method exploits a notion of desired object locomotion to guide the main policy during training. We also observe that DDPG-Sparse+SLDR fails for most tasks. A possible reason for this is that, despite its effectiveness, the proposed approach still requires a suitable RL algorithm to learn from SLDR together with sparse environmental rewards. DDPG on its own is less effective for this task. We find that HER-Dense performs worse than HER-Sparse. This result support previous observations that sparse rewards may be more beneficial for complex robot manipulation tasks compared to dense rewards (Andrychowicz et al., 2017; Plappert et al., 2018). Finally, we observe that CHER-Sparse fails in most tasks and cannot facilitate successful learning. This is somewhat expected given our particular set up, and a possible explanation is in order.

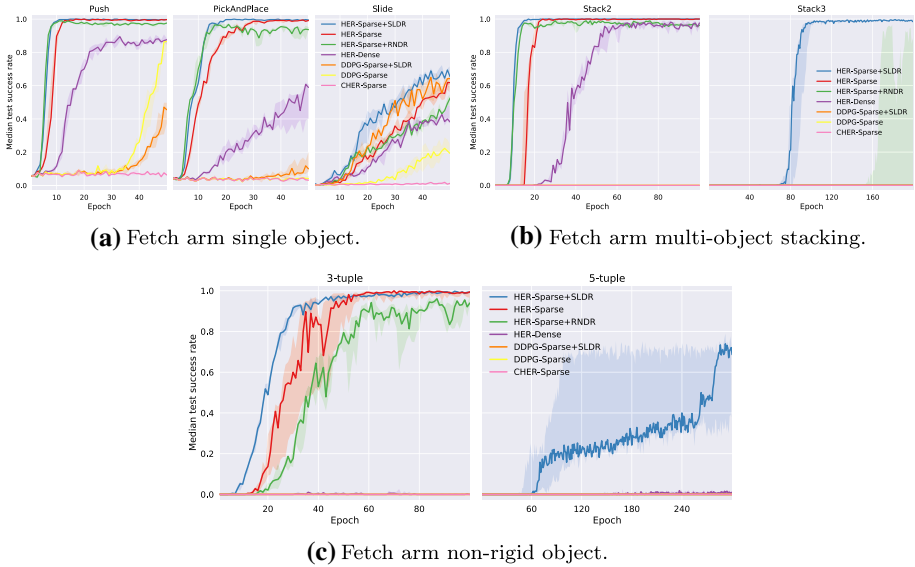


Fig. 4 Learning curves of comparison algorithms on environments using Fetch robotic arm

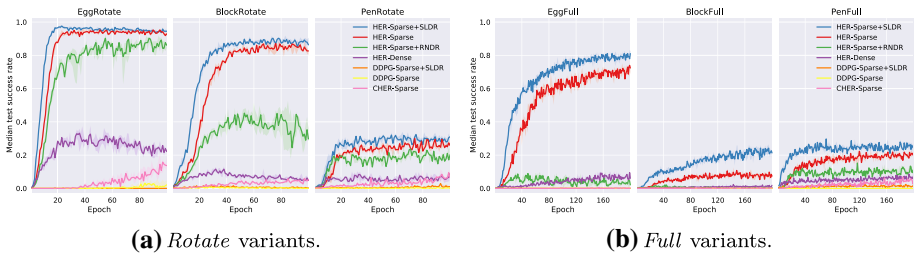


Fig. 5 Learning curves of comparison algorithms on environments using Shadow robotic hand

Sampling the replay buffer based on the proximity to true goals may work well for locomotion tasks because the distance between the robot gripper and the target is taken into account, and this distance is under direct control of the robot from the very first episode. On the other hand, in the manipulation tasks, the distance between the object and the target stays roughly constant in the early training episodes as the robot has not yet learned to interact with the object. Such a sampling technique prioritising the replays depending on proximity may produce biased batches that can potentially disrupt the learning process. For example, a random robot action causing the object to move away from the target would favour trajectories characterised by a lack of interaction between the robot and the object. Although we report some success on *EggRotate*, *BlockRotate* and *PenRotate* using CHER, this is much lower than the success observed when using HER-Sparse+SLDR and HER-Sparse.

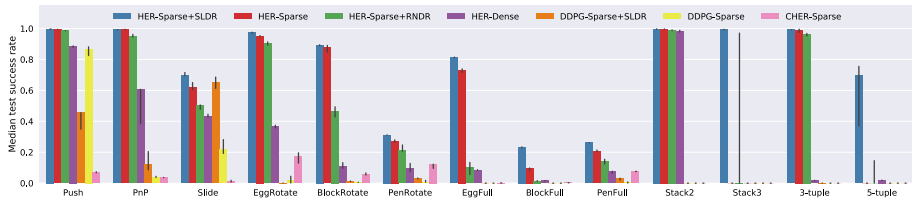


Fig. 6 Comparison of models with the best test success rate for all methods on all the environments

6.4 Fetch arm multiple object environments

For environments with N objects, we reuse the locomotion policies trained on the *Pick-AndPlace* task with single objects, and obtain an individual SLDR for each one of N objects. We train $N + 1$ action-value functions in total, i.e. one for each SLDR and one for the environmental sparse rewards. The manipulation policy is trained using the gradient in Eq. (11).

Inspired by Plappert et al. (2018), we randomly select between two initialisation settings for the training: (1) the targets are distributed on the table (i.e. an auxiliary task) and (2) the targets are stacked on top of each other (i.e. the original stacking task). Each initialisation setting is randomly selected with a probability of 0.5. We have observed that this initialisation strategy helps HER-based methods complete the stacking tasks. From Fig. 4b, we find that HER-Sparse+SLDR achieves better performance compared to HER-Sparse, HER-Sparse+RND and HER-Dense in the 2-object stacking task (*Stack2*), while other methods fail. On the more complex 3-object stacking task (*Stack3*), HER-Sparse+SLDR is the only algorithm to succeed. HER-Sparse+RND occasionally solves the *Stack3* task with fixed random seeds but the performance is unstable across different random seeds and multiple runs.

6.5 Fetch arm non-rigid object environments

The learning curves for 3-tuple and 5-tuple non-rigid object tasks are reported in Fig. 4c. Similarly to the multiple object environment, HER-Sparse+SLDR achieves better performance for the 3-tuple task compared to HER-Sparse and HER-Sparse+RND, while the other methods fail to complete the task. For the more complex 5-tuple task, only HER-Sparse+SLDR is able to succeed. Among the 4 pre-defined targets depicted in Fig. 3, HER-Sparse+SLDR can achieve 3 targets on average, and can accomplish all 4 targets in one instance, out of 5 runs with different random seeds.

6.6 Comparison Across the Best Models

Figure 6 summarises the performance of the models with the best test success rates for each one of the competing methods. We can see that the proposed HER-Sparse+SLDR achieves top performance compared to all other methods. Specifically, HER-Sparse+SLDR is the only algorithm that is able to steadily solve 3-object stacking (*Stack3*) and 5-tuple non-rigid object manipulation (*5-tuple*). Remarkably, these two tasks have the highest complexity among all the 13 tasks. The *Stack3* task includes multiple stages that require the

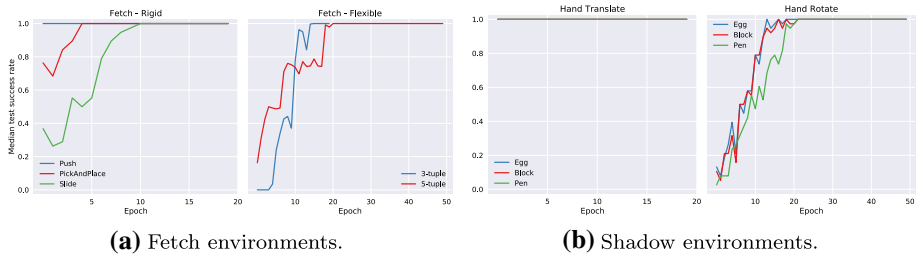


Fig. 7 Learning curves of for object locomotion policies

robot to pick and place multiple objects with different source and target locations in a fixed order; in the *5-tuple* task the object has the most complex dynamics. For these complex tasks, the proposed SLDR seems to be particularly beneficial. A possible reason is that, although the task is very complex, the objects are still able to learn good locomotion policies (see Fig. 7a) and the rewards learnt from locomotion policies provides critical feedback on how the object should be manipulated to complete the task. This type of object-based feedback is not utilised by other methods like HER and HER+RND. Our approach outperforms the runner-up by a large margin in the *Full* variants of Shadow's hand manipulation tasks (*EggFull*, *BlockFull* and *PenFull*), which feature complex state/action spaces and system dynamics. Finally, the proposed method consistently achieves better or similar performance than the runner-up in other simpler tasks.

7 Conclusion and discussion

In this paper, we address the problem of mastering robot manipulation through deep reinforcement learning using only sparse rewards. The rationale for the proposed methodology is that robot manipulation tasks can be seen of as inducing object locomotion. Based on this observation, we propose to firstly model the objects as independent entities that need to learn an optimal locomotion policy through interactions with a realistically simulated environment, then these policies are leveraged to improve the manipulation learning phase.

We believe that using SLDRs introduces significant advantages. First, SLDRs are generated artificially through a RL policy, hence require no human effort. Producing human demonstrations for complex tasks may prove difficult and/or costly to achieve without significant investments in human resources. For instance, it may be particularly difficult for a human to generate good demonstrations for tasks such as manipulating non-rigid objects with a single hand or with a robotic gripper. On the other hand, we have demonstrated that the locomotion policies can be easily learnt, even for complex tasks, purely in a virtual environment; e.g., in our studies, these policies have achieved 100% success rate on all tasks (e.g. see Fig. 7a, b). Furthermore, since the locomotion policy is learnt through RL, our proposed approach does not require task-specific domain knowledge and can be designed using only sparse rewards. Training the locomotion policies only requires the same sparse rewards provided by the environment hence the SLDRs produced through RL lead to high quality manipulation policies. This point has been supported by the empirical evidence obtained through experiments involving all 13 environments presented in this paper. As commonly observed in deep

RL approaches, the use of neural networks as a function approximators for policies and inverse dynamics functions may introduce convergence issues and lead to non-optimal policies, but despite these limitations the proposed methodology has been proved to be sufficiently reliable and competitive. The proposed approach is orthogonal to existing methods that use expert demonstrations, and combining them together would be an interesting direction to be explored in the future.

The performance of the proposed framework has been thoroughly examined on 13 robot manipulation environments of increasing complexity. These studies demonstrate that faster learning and higher success rate can be achieved through SLDRs compared to existing methods. In our experiments, SLDRs have enabled the robots to solve complex tasks, such as stacking 3 objects and manipulating non-rigid object with 5 tuples, whereas competing methods have failed. Remarkably, we have been able to outperform runner-up methods by a significant margin for complex Shadow's hand manipulation tasks. Although SLDRs are obtained using a physics engine, this requirement does not restrict the applicability of the proposed approach to situations where the manipulation is learnt using real robot as long as the locomotion policy can pre-learned realistically.

Several aspects will be investigated in follow-up work. We have noticed that when the interaction between the manipulating robot and the objects is very complex, the manipulation policy may be difficult to learn despite the fact that the locomotion policy is successfully learnt. For instance, in the case of the *5-tuple* task with Fetch arm, although the locomotion policy achieves a 100% success rate (as shown in Fig. 7a), the manipulation policy does not always completes the task (as shown in Figs. 4c and 6). In such cases, when the ideal object locomotion depends heavily on the robot, the benefit of the SLDs is reduced. Another limitation is given by our Assumption 2 (Sect. 4.2), which may not hold for some tasks. For example, for pen manipulation tasks with Shadow's hand, although the pen can rotate and translate itself to complete locomotion tasks (as shown in Fig. 7a), it is difficult for the robot to reproduce the same locomotion without dropping the pen. This issue can degrade the performance of the manipulation policy despite having obtained an optimal locomotion policy (see Figs. 5a, b and 6). A possible solution would be to train the manipulation policy and locomotion policy jointly, and check whether the robot can reproduce the object locomotion suggested by the locomotion policy; a notion of “reachability” of object locomotion could be used to regularise the locomotion policy and enforce $P(z_t|\mu_\theta) = P(z_t|v_\theta)$.

An important aspect to bear in mind is that our methodology requires the availability of a simulated environment for the application at hand. Nowadays, due to the well-documented sample inefficiency of most state-of-the-art, model-free DRL algorithms, such simulators are commonly used for training RL policies before deployment in the real world. Besides, creating physically realistic environments from existing 3D models using modern tools has become almost effortless. In this sense, the approach proposed in this work requires only a marginal amount of additional engineering once a simulator has been developed. For instance, using MuJoCo, setting up the object locomotion policies would only entail the removal of the robot from the environment and inclusion of the objects as “mocap” entities. In comparison with other approaches, such as those relying on human demonstrations, the additional effort required to enable SLDR is only minimal.

In this paper we have adopted DDPG as the main training algorithm due to its widely reported effectiveness in continuous control tasks. However, our framework is sufficiently general, and other algorithms may be suitable such as trust region policy optimisation (TRPO) (Schulman et al., 2015), proximal policy optimisation (PPO) (Schulman et al., 2017) and soft actor-critic (Haarnoja et al., 2018); analogously, model-based

methods (Chua et al., 2019; Zhang et al., 2019) could also provide feasible alternatives to be explored in future work.

Author Contributions OK proposed the idea, implemented the idea as a Python code, modified the gym environments for MuJoCo simulations, run the experiments, obtained the results and wrote the first draft of the manuscript. GM managed the project, provided the computational resources and revised the manuscript.

Availability of data and material Data used in the experiments has been generated through publicly available simulators. Related simulation files have been shared through the links given in the paper in order to fully reproduce the presented results.

Code availability Codes has been shared through the links given in the paper in order to fully reproduce the presented results.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International conference on machine learning*.
- Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2018). Learning dexterous in-hand manipulation. arXiv preprint [arXiv:1808.00177](https://arxiv.org/abs/1808.00177).
- Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. (2017). Hindsight experience replay. In *Advances in neural information processing systems* (pp. 5055–5065).
- Belter, D., Kopicki, M., Zurek, S., & Wyatt, J. (2014). Kinematically optimised predictions of object motion. In *2014 IEEE/RSJ international conference on intelligent robots and systems* (pp. 4422–4427). IEEE.
- Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., & Zhao, J. (2016). End to end learning for self-driving cars. arXiv preprint [arXiv:1604.07316](https://arxiv.org/abs/1604.07316).
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint [arXiv:1606.01540](https://arxiv.org/abs/1606.01540).
- Brown, D. S., Goo, W., Nagarajan, P., & Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on learning representations*.
- Burda, Y., Edwards, H., Storkey, A., & Klimov, O. (2018a). Exploration by random network distillation. arXiv preprint [arXiv:1810.12894](https://arxiv.org/abs/1810.12894).
- Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., & Efros, A. A. (2018b). Large-scale study of curiosity-driven learning. arXiv preprint [arXiv:1808.04355](https://arxiv.org/abs/1808.04355).
- Choi, S., Lee, K., & Songhwai, O. (2019). Robust learning from demonstrations with mixed qualities using leveraged gaussian processes. *IEEE Transactions on Robotics*, 35, 564–576.
- Chua, K., Calandra, R., McAllister, R., & Levine, S. (2019). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *International conference on machine learning*.
- Coumans, E., & Bai, Y. (2017). Pybullet, a python module for physics simulation in robotics, games and machine learning. <https://pybullet.org>.

- Duan, Y., Andrychowicz, M., Stadie, B. C., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., & Zaremba, W. (2017). One-shot imitation learning. In *Advances in neural information processing systems* (pp. 1087–1098).
- Fang, M., Zhou, T., Du, Y., Han, L., & Zhang, Z. (2019). Curriculum-guided hindsight experience replay.
- Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning* (pp. 49–58).
- Florensa, C., Held, D., Geng, X., & Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International conference on machine learning* (pp. 1514–1523).
- Fu, J., Levine, S., & Abbeel, P. (2016). One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International conference on intelligent robots and systems (IROS)* (pp. 4019–4026). IEEE.
- Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., & Darrell, T. (2018). Reinforcement learning from imperfect demonstrations. In *International conference on learning representations*.
- Grollman, D. H., & Billard, A. (2011). Donut as I Do: Learning from failed demonstrations. In *IEEE international conference on robotics and automation*.
- Gu, S., Holly, E., Lillicrap, T. P., & Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *International conference on robotics and automation* (pp. 3389–3396).
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*.
- Hester, T., et al. (2018). Deep q-learning from demonstrations. In *AAAI Conference on artificial intelligence* (pp. 3223–3230).
- Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems* (pp. 4565–4573).
- Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., & Abbeel, P. (2016). VIME: Variational information maximizing exploration. In *Advances in neural information processing systems*.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*. [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ international conference on intelligent robots and systems (IROS)*.
- Kopicki, M., Wyatt, J., & Stolkin, R. (2009). Prediction learning in robotic pushing manipulation. In *2009 International conference on advanced robotics* (pp. 1–6). IEEE.
- Kopicki, M., Zurek, S., Stolkin, R., Mörwald, T., & Wyatt, J. (2011). Learning to predict how rigid objects behave under simple manipulation. In *2011 IEEE international conference on robotics and automation* (pp. 5722–5729). IEEE.
- Kroemer, O., Niekum, S., & Konidaris, G. (2019). A review of robot learning for manipulation: Challenges, representations, and algorithms. *arXiv preprint* [arXiv:1907.03146](https://arxiv.org/abs/1907.03146).
- Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1), 1334–1373.
- Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., & Torralba, A. (2018). Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint* [arXiv:1810.01566](https://arxiv.org/abs/1810.01566).
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR*. [arXiv:1509.02971](https://arxiv.org/abs/1509.02971).
- Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).
- Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018a). Overcoming exploration in reinforcement learning with demonstrations. In *International conference on robotics and automation* (pp. 6292–6299).
- Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., & Levine, S. (2018b). Visual reinforcement learning with imagined goals. In *Advances in neural information processing systems* (pp. 9209–9220).
- Ng, A. Y., & Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *International conference on machine learning* (pp. 663–670).
- Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *IEEE Conference on computer vision and pattern recognition workshops* (pp. 16–17).
- Plappert, M., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*. [arXiv:1802.09464](https://arxiv.org/abs/1802.09464).

- Pomerleau, D. (1988). ALVINN: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems* (pp. 305–313).
- Popov, I., Heess, N., Lillicrap, T. P., Hafner, R., Barth-Maron, G., Vecerik, M., Lampe, T., Tassa, Y., Erez, T., & Riedmiller, M. A. (2017). Data-efficient deep reinforcement learning for dexterous manipulation. *CoRR*. [arXiv:1704.03073](https://arxiv.org/abs/1704.03073).
- Ratliff, N. D., Bagnell, J. A., & Srinivasa, S. S. (2007). Imitation learning for locomotion and manipulation. In *International conference on humanoid robots* (pp. 392–397).
- Reddy, S., Dragan, A. D., & Levine, S. (2019). SQL: Imitation learning via regularized behavioral cloning. *arXiv preprint* [arXiv:1905.11108](https://arxiv.org/abs/1905.11108).
- Riedmiller, M. A., et al. (2018). Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning* (pp. 4341–4350).
- Ross, S., Gordon, G. J., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *International conference on artificial intelligence and statistics* (pp. 627–635).
- Sasaki, F., Yohira, T., & Kawaguchi, A. (2019). Sample efficient imitation learning for continuous control. In *International conference on learning representations*.
- Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., & Gelly, S. (2018). Episodic curiosity through reachability. *arXiv preprint* [arXiv:1810.02274](https://arxiv.org/abs/1810.02274).
- Schaul, T., Horgan, D., Gregor, K., & Silver, D. (2015). Universal value function approximators. In *Proceedings of the 32nd international conference on machine learning, ICML 2015, Lille, France, 6–11 July 2015* (pp. 1312–1320).
- Schmidhuber, J. (1991). Curious model-building control systems. In *IEEE international joint conference on neural networks* (pp. 1458–1463). IEEE.
- Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint* [arXiv:1707.06347](https://arxiv.org/abs/1707.06347).
- Shiari, K., Messias, J., & Whiteson, S. (2016). Inverse reinforcement learning from failure. In *International conference on autonomous agents & multiagent systems*.
- Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. A. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning* (pp. 387–395).
- Silver, D., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.
- Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., & Fergus, R. (2018). Intrinsic motivation and automatic curricula via asymmetric self-play. In *International conference on learning representations*.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.
- Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems* (pp. 5026–5033). IEEE.
- Vecerik, M., et al. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR*. [arXiv:1707.08817](https://arxiv.org/abs/1707.08817).
- Wang, Z., Merel, J., Reed, S., Wayne, G., de Freitas, N., & Heess, N. (2017). Robust imitation of diverse behaviors. *arXiv preprint* [arXiv:1707.02747](https://arxiv.org/abs/1707.02747).
- Watkins, C. J. C. H., & Dayan, P. (1992). Technical note q-learning. *Machine Learning*, 8, 279–292.
- Xu, H., Gao, Y., Yu, F., & Darrell, T. (2017). End-to-end learning of driving models from large-scale video datasets. In *IEEE Conference on computer vision and pattern recognition*.
- Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M., & Levine, S. (2019). SOLAR: deep structured representations for model-based reinforcement learning. In *International conference on machine learning*.
- Zheng, J., Liu, S., & Ni, L. M. (2014). Robust Bayesian inverse reinforcement learning with sparse behavior noise. In *AAAI conference on artificial intelligence*.
- Zhu, H., Gupta, A., Rajeswaran, A., Levine, S., & Kumar, V. (2018). Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. *arXiv preprint* [arXiv:1810.06045](https://arxiv.org/abs/1810.06045).